TDB – NLA

Numerical Linear Algebra

Self reading: Introduction, dense matrices

Maya Neytcheva, TDB, Feb-March 2021

When talking about the solution of a linear system of equations:

- computational complexity - computer demands (computing time and memory consumption)
- robustness wrt to (problem, discretization and method) parameters

---

- numerical efficiency (later, for iterative methods - number of iterations)
- parallelization aspects, HPC flavour

Before discussing sparse matrices...

we are going to look first at dense matrices...
because these are easier.

Large dense matrices

GOAL: get a global overview of issues related to direct solution methods:

- Gauss elimination
- LU factorization, Cholesky factorization
- stability, pivoting, errors;
- complexity
- effect of the dense/sparse structure on the performance

Large dense matrices

An idea what matrix dimensions might have been considered very large for a dense, direct matrix computation through the years:

n	Year	Source
20	1950	Wilkinson
200	1965	Forsythe&Moler
2000	1980	LINPACK
20000	1995	LAPACK
$> 200\,000$	some years ago	(Umeå)

J. Wilkinson, The algebraic eigenvalue problem, 1965
G. Forsythe& C. Moler, Computer solutions of linear algebraic systems, 1967.

Computational complexity issues: Cramer's rule

$$Ax = \boldsymbol{b}, \quad A(n \times n), \quad det(A) \neq 0$$

$$\begin{bmatrix} a_{11} & \cdots & a_{1,i-1} & a_{1,i} & a_{1,i+1} & \cdots & a_{1,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{i1} & \cdots & a_{i,i-1} & a_{i,i} & a_{i,i+1} & \cdots & a_{i,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & \cdots & a_{n,i-1} & a_{n,i} & a_{n,i+1} & \cdots & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ \cdots \\ x_i \\ \cdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \cdots \\ b_i \\ \cdots \\ b_n \end{bmatrix}$$

Computational complexity issues: Cramer's rule

$$x_i = \frac{1}{det(A)} \left(\begin{bmatrix} a_{11} & \cdots & a_{1,i-1} & b_1 & a_{1,i+1} & \cdots & a_{1,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{i1} & \cdots & a_{i,i-1} & b_i & a_{i,i+1} & \cdots & a_{i,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & \cdots & a_{n,i-1} & b_n & a_{n,i+1} & \cdots & a_{n,n} \end{bmatrix} \right)$$

Computational complexity issues

Consider products of n elements of A,

$$a_{1,\alpha_1}, a_{2,\alpha_2}, \cdots, a_{n,\alpha_n},$$

where $\alpha_1, \alpha_2, \cdots, \alpha_n$ is a permutation of $1, 2, \cdots, n$.
The number of all these products is $\boxed{n!}$.

$$det(A) = \sum_{i=1} n! \prod_{j=1}^{n} (-1)^{\gamma} a_{j,\alpha_j},$$

thus, the computational complexity to solve the system is $n!$.
To be more precise: $(n+1)(n!) = (n+1)!$ multiplications and $(n+1)(n!) = (n+1)!$ additions.

Gauss elimination/LU factorization: $A(m,n)$

```
for k = 1, 2 ··· m − 1
  d = 1/a_kk^(k)
  for i = k + 1, ··· m
    ℓ_ik^(k) = −a_ik^(k) d
    for j = k + 1, ··· n
      a_ij^(k) = a_ij^(k) + ℓ_ik a_kj^(k)
    end
  end
end
```

The operational count for the LU factorization can be obtained by integrating the loops:

$$Flops_{LU} = \int_{1}^{m-1}\int_{k}^{m}\int_{k}^{n} d_j d_i d_k \approx n^3/3 \ (m = n)$$

Computational complexity issues: computing $det(A)$

Method	Multiplications	Additions
Gaussian Elimination	$\frac{1}{3}n^3 + n^2 - \frac{1}{3}n$	$\frac{1}{3}n^3 + \frac{1}{2}n^2 - \frac{5}{6}n$
Gauss-Jordan Elimination	$\frac{1}{3}n^3 + n^2 - \frac{5}{2}n + 2$	$\frac{1}{3}n^3 - \frac{3}{2}n^2 + 1$
Cramer's Rule	$n!$	$n!$

Computational complexity issues: Cramer against Gauss

A comparison of the amount of time to solve $Ax = b$ on a Cray J90. The Cray J90 performs one trillion operations per second (one teraflop).

n	Gaussian Elimination	Cramer's Rule
2	6×10^{-12} secs	6×10^{-12} secs
3	1.7×10^{-11} secs	2.4×10^{-11} secs
4	3.6×10^{-11} secs	1.2×10^{-10} secs
5	6.5×10^{-11} secs	7.2×10^{-10} secs
6	1.06×10^{-10} secs	5.04×10^{-09} secs
10	4.3×10^{-10} secs	3.99168×10^{-05} secs
20	3.06×10^{-9} secs	1.622 years
100	3.433×10^{-7} secs	2.9889×10^{138} centuries
1000	3.3433×10^{-4} secs	

Computational complexity issues: hardware development

Top500, November 2021, no 1:
Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Fujitsu
RIKEN Center for Computational Science

- performance of 537.212 petaflops on High Performance Linpack,
- Tofu interconnect D
- 7,630,848 cores
- 5,087,232 GB memory

Factorials...

In 2001, the value of 1000! was currently too large to be stored as a single number in the memory of a computer.
(*Computational Science: Tools for a Changing World* by R.A. Tapia, C. Lanius, 2001, Rice.)

The scientific calculator in Windows XP is able to calculate factorials up to at least 100000!.
(look-up tables)

Dense LU remains an active field of research:

"Dense Matrix Factorization of Linear Complexity for Impedance Extraction of Large-Scale 3-D Integrated Circuits"
Wenwen Chai, Dan Jiao School of Electrical and Computer Engineering, Purdue University, IEEE Xplore, July 2010

Abstract: A fast LU factorization of linear complexity is developed to directly solve a dense system of linear equations for the interconnect extraction of any arbitrary shaped 3-D structure embedded in inhomogeneous materials. The proposed solver successfully factorizes dense matrices that involve more than one million unknowns in fast CPU run time and modest memory consumption. Comparisons with state-of-the-art integral equation- based interconnect extraction tools have demonstrated its clear advantages.

Dense LU remains an active field of research:

Programming parallel dense matrix factorizations with look-ahead and OpenMP
Sandra Catalán, Adrián Castelló, Francisco D. Igual, Rafael Rodríguez-Sánchez Enrique S. Quintana-Ortí
Cluster Computing, 23, 359–375(2020)
We investigate a parallelization strategy for dense matrix factorization (DMF) algorithms, using OpenMP, that departs from the legacy (or conventional) solution ...

Stability of Gauss elimination

... is unstable !

Factorizing symmetric positive definite matrices

Factorize $A = LL^T$, L – lower-triangular
Cholesky factorization

Factorizing symmetric marices

The mathematician after whom the Cholesky factorisation is named.

Major Andre-Louis Cholesky (1875-1918)

Born in France, worked in the Geodesic section of the Geographic service to the French army's artillery branch.
At this time the system of triangulation used in France, and based on the meridian line of Paris, was being revised; new methods were needed in order to facilitate what was not yet a quick or convenient process.
Cholesky invented computation procedures based on the method of least squares, for the solution of certain data-fitting problems in geodesy, to be put into practice in his triangulation of the French and British parts of Crete, and in his work in Algeria and Tunisia.
His mathematical work was posthumously published on his behalf in 1924 by a fellow officer, Benoit.

Cholesky factorization ...

```
% Maya's version of Cholesky - to compare execution time
% ----------------------------------------------------------------
function [U]=my_chol(A)
A = triu(A);
n = size(A,1);
for k=1:n,
    A(1:k-1,k) = A(1:k-1,1:k-1)'\A(1:k-1,k);
    A(k,k) = sqrt(A(k,k) - A(1:k-1,k)'*A(1:k-1,k));
end
U = triu(A);
return
```

Cholesky factorization ...

size(A)	chol Matlab	chol mine	Ratio		
10	0.000292		0.004360		14.9315
50	0.000183	0.6267	0.002697	0.6186	14.7377
100	0.000327	1.7869	0.002305	0.8547	7.0489
500	0.002132	6.5199	0.264100	114.5770	123.8724
1000	0.008465	3.9705	0.970080	3.6732	114.5987
5000	0.583840	68.9711	161.698800	166.6860	276.9573

Outer Product Cholesky

for $k = 1 : n$

$A(k,k) = \sqrt{A(k,k)}$

$A(k+1:n,k) = A(k+1:n,k) - A(n:k,k-1)/A(k,k)$

for $j = k+1 : n$

$A(j:n,j) = A(j:n,j) - A(j:n,j)A(j,k)$

end

end

Example of implementing Cholesky factorization

```
for k=1:n
    xeuitb(A(1:k-1,k),A(1:k-1,1:k-1),A(1:k-1,k))
    A(k,k) = sqrt(A(k,k) - A(1:k-1,k)^T*A(1:k-1,k))
end
```

Computes U (which overwrites A).

BLAS `xeuitb(X,U,B)` computes $X = U^{-1}B$